NUMPY BASIC:

1. Import

```
import numpy as np
```

2. Array Creation

- From list:

```
a = np.array([1, 2, 3])           # 1D
b = np.array([[1,2],[3,4]])       # 2D
```

- Special arrays:

```
np.zeros((2,3))     # 2x3 all zeros
np.ones((3,3))      # 3x3 all ones
np.eye(3)           # Identity matrix
np.arange(0,10,2)   # [0,2,4,6,8]
np.linspace(0,1,5)  # 5 points between 0 and 1
```

3. Properties

```
a.shape      # dimensions (rows, cols)
a.ndim       # number of dimensions
a.size       # total elements
a.dtype      # data type
```

---

4. Indexing & Slicing

```
arr = np.arange(10)    # [0 1 2 3 4 5 6 7 8 9]
arr[2]         # 2
arr[2:7]       # [2 3 4 5 6]
arr[::-1]      # reverse

mat = np.array([[1,2,3],[4,5,6]])
mat[0,1]       # 2
mat[:,1]       # column [2 5]
```

---

5. Operations

- Elementwise:

```
x = np.array([1,2,3])
y = np.array([10,20,30])
x + y    # [11 22 33]
x * y    # [10 40 90]
```

- Broadcasting:

```
M = np.ones((3,3))
v = np.array([1,2,3])
M + v    # adds row-wise
```

---

6. Aggregate Functions

```
a = np.arange(1,7).reshape(2,3)   # [[1 2 3],[4 5 6]]
a.sum()            # 21
a.mean(axis=0)     # [2.5 3.5 4.5] (col-wise)
a.min(axis=1)      # [1 4] (row-wise)
```

---

7. Linear Algebra

```
A = np.array([[1,2],[3,4]])
B = np.array([[5,6],[7,8]])
np.dot(A,B)          # matrix multiplication
np.transpose(A)      # transpose
np.linalg.inv(A)     # inverse
np.linalg.eig(A)     # eigenvalues & eigenvectors
```

---

8. Random

```
np.random.rand(2,3)     # uniform [0,1)
np.random.randn(2,3)    # normal dist
np.random.randint(1,10,(2,2))  # integers
```

2. Array Creation

```
np.array([1,2,3])            # create array
np.zeros((2,3))              # zeros
np.ones((2,3))               # ones
np.empty((3,3))              # uninitialized
np.full((2,2), 7)            # constant fill
np.arange(0,10,2)            # step sequence
np.linspace(0,1,5)           # evenly spaced
np.eye(3)                    # identity
np.random.rand(2,2)           # random (0,1)
```

3. Shape / Reshape

```
arr.shape                     # shape
arr.ndim                      # dimensions
arr.size                      # number of elements
arr.reshape(3,2)              # reshape
arr.ravel()                   # flatten
arr.T                         # transpose
```

4. Mathematical / Aggregate

```
np.sum(arr)                   # sum
np.mean(arr)                  # average
np.median(arr)                # median
np.std(arr)                   # standard deviation
np.var(arr)                   # variance
np.min(arr), np.max(arr)      # min, max
np.argmin(arr), np.argmax(arr) # index of min, max
np.cumsum(arr)                # cumulative sum
np.cumprod(arr)               # cumulative product
```

5. Element-wise Ops

```
np.sqrt(arr)                  # square root
np.exp(arr)                   # exponential
np.log(arr)                   # natural log
np.abs(arr)                   # absolute value
np.round(arr, 2)              # rounding
```

6. Sorting & Searching

```
np.sort(arr)                  # sort copy
arr.sort()                    # in-place sort
np.argsort(arr)               # indices for sort
np.unique(arr)                # unique elements
np.where(arr > 5)             # condition indices
np.extract(arr % 2 == 0, arr) # values meeting condition
```

7. Linear Algebra

```
np.dot(A,B)                  # matrix multiply
np.matmul(A,B)               # same as dot
np.linalg.inv(A)             # inverse
np.linalg.det(A)             # determinant
np.linalg.eig(A)             # eigenvalues, eigenvectors
```

8. File I/O

```
np.save("array.npy", arr)    # save binary
np.load("array.npy")         # load binary
np.savetxt("data.csv", arr, delimiter=",")
np.loadtxt("data.csv", delimiter=",")
```

Pandas Basic:

1. Import

```
import pandas as pd
```

---

2. Core Data Structures

- **Series** → 1D labeled array.
- **DataFrame** → 2D labeled table.

```
s = pd.Series([10, 20, 30], index=["a","b","c"])
df = pd.DataFrame({"Name":["A","B","C"], "Age":[20,25,30]})
```

---

3. Inspecting Data

```
df.head()          # first 5 rows
df.tail(3)         # last 3 rows
df.shape           # (rows, cols)
df.info()          # column info
df.describe()      # statistics for numeric cols
df.columns         # column names
df.index           # row index
```

---

4. Selection

```
df["Age"]                  # column
df[["Name","Age"]]         # multiple columns
df.loc[0]                  # by label
df.iloc[0]                 # by position
df.loc[0,"Name"]           # single value
```

---

5. Filtering

```
df[df["Age"] > 22]         # condition
df[(df["Age"]>20) & (df["Age"]<30)]
```

---

6. Adding / Modifying

```
df["Salary"] = [100,200,300]   # new col
df.loc[0,"Age"] = 21           # update value
df.drop("Salary", axis=1)      # drop col
df.drop(0, axis=0)             # drop row
```

7. Missing Data

```
df.isnull()            # check NaN
df.dropna()            # drop missing
df.fillna(0)           # replace missing
```

8. Aggregation & Grouping

```
df["Age"].mean()                 # mean
df.groupby("Age").size()         # count per group
df.groupby("Age")["Salary"].sum()
```

9. Sorting

```
df.sort_values("Age")            # by column
df.sort_index()                  # by index
```

10. File I/O

```
df = pd.read_csv("data.csv")
df.to_csv("out.csv", index=False)

df = pd.read_excel("data.xlsx")
df.to_excel("out.xlsx", index=False)
```

1. Information & Overview

```
df.head(n)            # first n rows
df.tail(n)            # last n rows
df.info()             # column summary
df.describe()         # stats for numeric cols
df.dtypes             # data types
df.memory_usage()     # memory usage
df.shape              # (rows, cols)
```

2. Selection & Indexing

```
df["col"]                  # single col
df[["col1","col2"]]        # multiple cols
df.loc[2,"col1"]           # by label
df.iloc[2,0]               # by position
df.at[2,"col1"]            # fast scalar
df.iat[2,0]                # fast scalar
```

3. Filtering & Boolean Indexing

```
df[df["Age"] > 25]
df[(df["Age"]>20) & (df["Salary"]<50000)]
df.query("Age > 20 and Salary < 50000")
```

4. Adding, Modifying, Dropping

```
df["NewCol"] = df["Age"] * 2
df.rename(columns={"Age":"Years"}, inplace=True)
df.drop("NewCol", axis=1, inplace=True)
df.drop([0,1], axis=0)               # drop rows
```

5. Handling Missing Data

```
df.isnull().sum()        # count NaN
df.dropna()              # drop rows with NaN
df.fillna(0)             # replace NaN with 0
df.fillna(df.mean())     # replace with mean
df.interpolate()         # linear interpolation
```

6. Aggregation & Grouping

```
df["Salary"].sum()
df["Age"].mean()
df.groupby("Dept")["Salary"].mean()
df.groupby(["Dept","Gender"]).size()
df.pivot_table(values="Salary", index="Dept", columns="Gender")
```

7. Sorting

```
df.sort_values("Age")
df.sort_values(["Dept","Salary"], ascending=[True,False])
df.sort_index()
```

8. Combining DataFrames

```
pd.concat([df1,df2], axis=0)     # stack rows
pd.concat([df1,df2], axis=1)     # add cols
pd.merge(df1,df2, on="id")       # SQL-style join
pd.merge(df1,df2, how="outer", on="id")
```

9. Useful Column/Row Ops

```
df["col"].unique()              # unique values
df["col"].nunique()             # count uniques
df["col"].value_counts()        # frequency
df.apply(np.sqrt)               # apply function
df["col"].map(lambda x: x*2)    # elementwise
df.applymap(str.upper)          # elementwise for df
```

10. Datetime Functions

```
df["Date"] = pd.to_datetime(df["Date"])
df["Year"] = df["Date"].dt.year
df["Month"] = df["Date"].dt.month
df["Day"] = df["Date"].dt.day
df.set_index("Date", inplace=True)
df.resample("M").mean()         # monthly mean
```

11. Export / Import

```
df.to_csv("file.csv", index=False)
df.to_excel("file.xlsx", index=False)
df.to_json("file.json")
pd.read_sql("SELECT * FROM table", conn)
```

Matplotlib Basic:

1. Import

```
import matplotlib.pyplot as plt
```

---

2. Line Plot

```
x = [1,2,3,4]
y = [2,4,6,8]

plt.plot(x, y, label="Line", color="r", linestyle="--", marker="o")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("Line Plot Example")
plt.legend()
plt.grid(True)
plt.show()
```

---

3. Scatter Plot

```
plt.scatter(x, y, color="g", marker="x")
plt.title("Scatter Plot")
plt.show()
```

---

4. Bar Chart

```
names = ["A","B","C"]
values = [4,7,3]

plt.bar(names, values, color="purple")
plt.title("Bar Chart")
plt.show()
```

---

5. Histogram

```
import numpy as np
data = np.random.randn(1000)

plt.hist(data, bins=30, color="skyblue", edgecolor="black")
plt.title("Histogram")
plt.show()
```

---

6. Pie Chart

```
sizes = [40, 30, 20, 10]
labels = ["A","B","C","D"]

plt.pie(sizes, labels=labels, autopct="%1.1f%%", startangle=90)
plt.title("Pie Chart")
plt.show()
```

---

7. Subplots

```
x = np.linspace(0,10,100)
y = np.sin(x)

plt.subplot(2,1,1)       # (rows, cols, index)
plt.plot(x, y, "r")
plt.title("Sine")

plt.subplot(2,1,2)
plt.plot(x, np.cos(x), "b")
plt.title("Cosine")

plt.tight_layout()
plt.show()
```

---

8. Customization

```
plt.plot(x,y)
plt.xlim(0,10)             # set x range
plt.ylim(-1,1)             # set y range
plt.xticks([0,5,10])       # custom ticks
plt.yticks([-1,0,1])
plt.show()
```

1. Figure & Axes (Object-Oriented Interface)

```
fig, ax = plt.subplots()                 # create figure + axes
ax.plot([1,2,3],[4,5,6])                 # plot on axes
ax.set_title("Using OO Interface")
ax.set_xlabel("X")
ax.set_ylabel("Y")
plt.show()
```

👉 `plt.plot()` is quick, but `fig, ax` is preferred for multiple plots.

---

2. Multiple Figures

```
plt.figure(1)
plt.plot([1,2,3],[1,4,9])

plt.figure(2)
plt.plot([1,2,3],[1,2,3])
plt.show()
```

---

3. Styles

```
plt.style.available        # list styles
plt.style.use("ggplot")    # apply style
```

---

4. Saving Plots

```
plt.plot([1,2,3],[4,5,6])
plt.savefig("plot.png")                # save as PNG
plt.savefig("plot.pdf", dpi=300)       # high-res PDF
```

---

5. Annotations

```
x = [1,2,3]; y = [2,4,6]
plt.plot(x,y)
plt.annotate("Point", xy=(2,4), xytext=(2.5,4.5),
             arrowprops=dict(facecolor="black", shrink=0.05))
plt.show()
```

---

6. Fill Between

```
x = np.linspace(0,10,100)
y1, y2 = np.sin(x), np.sin(x)+0.5

plt.fill_between(x, y1, y2, color="lightblue")
plt.plot(x,y1,"r",x,y2,"b")
plt.show()
```

---

7. Stacked Bar

```
A = [3,2,4]
B = [1,2,3]
labels = ["X","Y","Z"]

plt.bar(labels, A, label="A")
plt.bar(labels, B, bottom=A, label="B")
plt.legend()
plt.show()
```

---

8. Boxplot & Violin Plot

```
data = np.random.randn(100)

plt.boxplot(data)
plt.title("Boxplot")
plt.show()
```

---

9. 3D Plotting

```
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure()
ax = fig.add_subplot(111, projection="3d")
x = np.random.rand(100)
y = np.random.rand(100)
z = np.random.rand(100)

ax.scatter(x,y,z)
plt.show()
```

---

10. Heatmap (with `imshow`)

```
matrix = np.random.rand(5,5)
plt.imshow(matrix, cmap="viridis", interpolation="nearest")
plt.colorbar()
plt.show()
```

SeaBorn Basic:

1. Import & Style

```
import seaborn as sns
import matplotlib.pyplot as plt

sns.set_style("darkgrid")    # other: white, whitegrid, dark
```

---

2. Example Dataset

```
tips = sns.load_dataset("tips")
tips.head()
```

👉 Seaborn works **directly with Pandas DataFrames**.

---

3. Scatter Plot

```
sns.scatterplot(x="total_bill", y="tip", data=tips, hue="sex", style="time",
size="size")
plt.title("Scatter Plot")
plt.show()
```

---

4. Line Plot

```
sns.lineplot(x="size", y="tip", data=tips, hue="sex", marker="o")
```

---

5. Bar Plot

```
sns.barplot(x="day", y="total_bill", data=tips, estimator=sum, ci=None)
```

---

6. Count Plot

```
sns.countplot(x="day", data=tips, hue="sex")    # frequency of categories
```

---

7. Histogram & KDE

```
sns.histplot(tips["total_bill"], bins=20, kde=True)
```

---

8. Box Plot

```
sns.boxplot(x="day", y="total_bill", data=tips, hue="sex")
```

---

9. Violin Plot

```
sns.violinplot(x="day", y="total_bill", data=tips, inner="quartile")
```

---

10. Heatmap

```
corr = tips.corr(numeric_only=True)
sns.heatmap(corr, annot=True, cmap="coolwarm")
```

---

11. Pair Plot

```
sns.pairplot(tips, hue="sex")
```

OOPS Basic:

1. Class & Object

- **Class** → Blueprint (defines attributes + methods).
- **Object** → Instance of class.

```
class Car:
    def __init__(self, brand, model):   # constructor
        self.brand = brand
        self.model = model

    def info(self):                     # method
        return f"{self.brand} {self.model}"

c1 = Car("Toyota","Fortuner")           # object
print(c1.info())
```

---

2. Encapsulation

- Restrict direct access to variables.
- Use `_var` (protected) or `__var` (private).

```
class Student:
    def __init__(self, name, marks):
        self.__name = name     # private
        self._marks = marks    # protected

    def get_name(self):
        return self.__name
```

---

3. Inheritance

- Derive a class from another.

```
class Animal:
    def speak(self):
        print("Some sound")

class Dog(Animal):                # single inheritance
    def speak(self):
        print("Bark")
```

```
class Puppy(Dog):               # multilevel
    pass

d = Dog()
d.speak()
```

4. Polymorphism

- **Method Overriding** → Child class redefines parent method.
- **Operator Overloading** → Define special methods like `__add__`.

```
class Complex:
    def __init__(self, r, i):
        self.r, self.i = r, i

    def __add__(self, other):   # overloading +
        return Complex(self.r+other.r, self.i+other.i)

c1 = Complex(1,2); c2 = Complex(3,4)
c3 = c1 + c2
print(c3.r, c3.i)
```

5. Abstraction

- Hide implementation, expose only interface.
- Achieved with **abstract base class** (`abc` module).

```
from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def area(self): pass

class Circle(Shape):
    def __init__(self,r): self.r = r
    def area(self): return 3.14 * self.r * self.r
```

6. Important Keywords

- `self` → refers to the current object.
- `__init__` → constructor.
- `__str__` → string representation.
- `isinstance(obj, Class)` → check object type.
- `issubclass(C1, C2)` → check inheritance.

⚡ Quick Recall

- **Class/Object** → blueprint + instance
- **Encapsulation** → `_protected`, `__private`
- **Inheritance** → reuse & extend functionality
- **Polymorphism** → same name, different behavior (`overriding`, `overloading`)
- **Abstraction** → hide details, use abstract class